Tutorial: Creating Platform Games

Last changed: September 4, 2004
Uses: version 6.0, advanced mode
Level: Intermediate

Platform games are very common, in particular on devices like the Game Boy. In a platform game you look at the scene from the side. The player normally controls a character that walks around in the world. This world consists of platforms. The player can walk on these platforms, jump or drop from one platform to the other, use ladders or ropes to get to different places, etc. On the platforms there are objects to collect, enemies to avoid or kill (often either by shooting them or by jumping on top of them), switches to press to open passages, etc. Also the player normally requires skill to jump over dangerous areas. In some platform games you see the whole level at once, but in most you see only a part around the character. In such a case, finding your way around becomes an additional challenge.

Creating a good platform game is not trivial, also not with *Game Maker*. There are three important aspects:

- Creating a natural motion for the character.
- Creating enough variation in monsters, background, etc.
- Carefully designing the levels such that they are fun to play and get increasingly difficult.

In this tutorial I will give some advice and tricks about how to create platform games using *Game Maker.* The tutorial is accompanied by a number of demo games. These are not full games. They consist of just one level to demonstrate some particular aspect. You can use them as a basis for your own platform games.

The basics

We start with the most simple platform game. You can find it in the file `platform_1.gm6`. In each platform game there are two basic objects: the character that is controlled by the player, and a block object that is used for the floors (platforms) the player can walk on. The same block is often used for the walls that the player cannot pass. We need two sprites: one for the character and one for the block. For the character we use a simple ball. For the block we use a (non-transparent) black square. We create two objects. The block object is simply a solid object that has no events or actions. It simply sits there. The character object is a lot more complicated.

Motion

The crucial aspect we treat in this first section is how to define the motion of the character. The problem is that the character must walk on top of the floors. It must not intersect the floor. If the character jumps or falls off a platform it must land correctly on the next platform. There are a number of different ways in which the character can walk, jump, and fall. Different platform games use different modes. Normally we just use three keys to control the motion. The left arrow key should move the character to the left, the right arrow key should move it to the right, and the up key or the space key makes it jump.

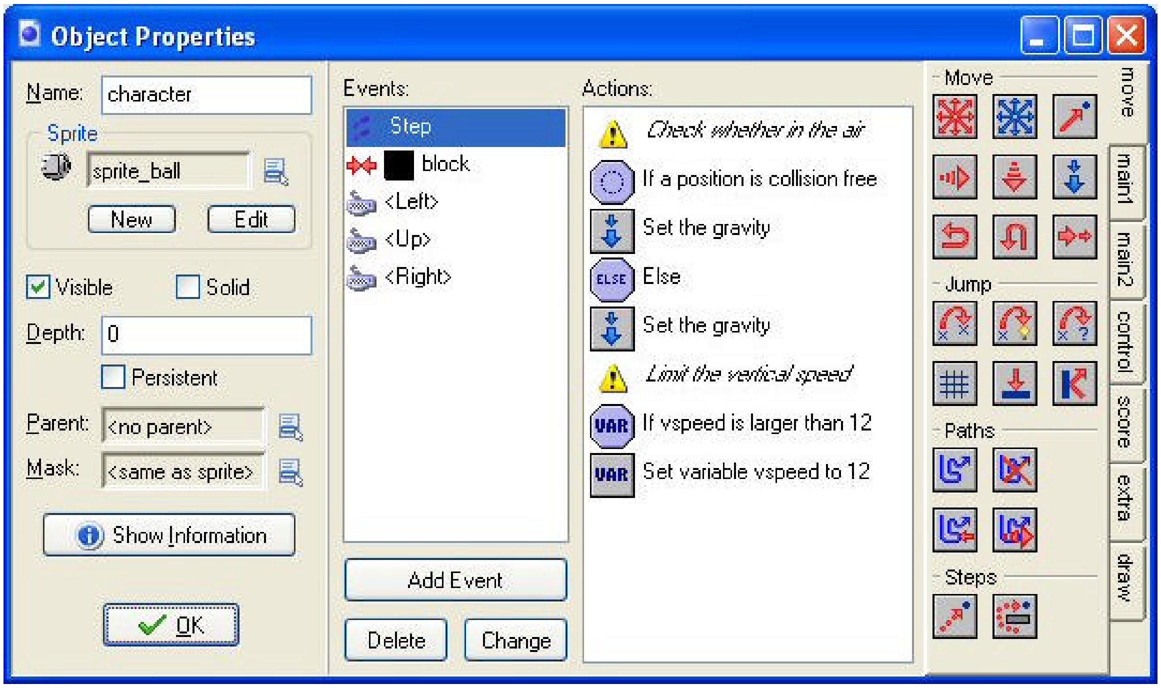
Let us first consider the left and right motion. The first choice to make it whether the player can only change its direction of motion while on a platform or also in the air while jumping or falling. Even though the second option is not natural (it is rather difficult to start moving left while you are falling down) we decide to go for the first option, that is, we allow horizontal motion wherever the character is. This tends to lead to nicer game play and is actually also easier to implement.

The second choice is whether the motion has constant speed or whether it accelerates when you keep the key pressed. For simplicity reasons we opt for the first choice. Allowing for acceleration though normally gives nicer game play: the player must for example start a run at a distance to jump over a wide hole.

As you should know there are different ways to let a character move. We can set a speed of motion or we can simply move the character directly. In platform games it is normally the easiest to let the vertical motion be done automatically (as we will see below) but to do the horizontal motion ourselves. This is rather easy. In the keyboard event for the left arrow key we check whether the position at relative position (-4,0) is free. If so we let the character jump to that position. We treat the right arrow key in a similar way. See the enclosed example game.

Jumping

Next we need the vertical motion. This is more difficult. To let the character fall down we can use gravity. But it should stop moving when we hit the floor. Also, you normally want a maximal falling speed, otherwise the character will move too fast. (This is both not very pleasing but it can also cause problems in the implementation. E.g. the character might fall through a floor if it moves too fast.) To solve this problem, in the step event of the character we check whether the position just below the character is collision free. If so, the character is in the air and we set the gravity to a positive value. Otherwise we set it to 0. We also check the variable `vspeed` which indicates the vertical speed. If it is larger to 12 we set it back to 12. In this way we limit the vertical speed to 12. So the event looks something like this:



Name: character

Sprite: sprite_ball

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:
- Step
- block
- <Left>
- <Up>
- <Right>

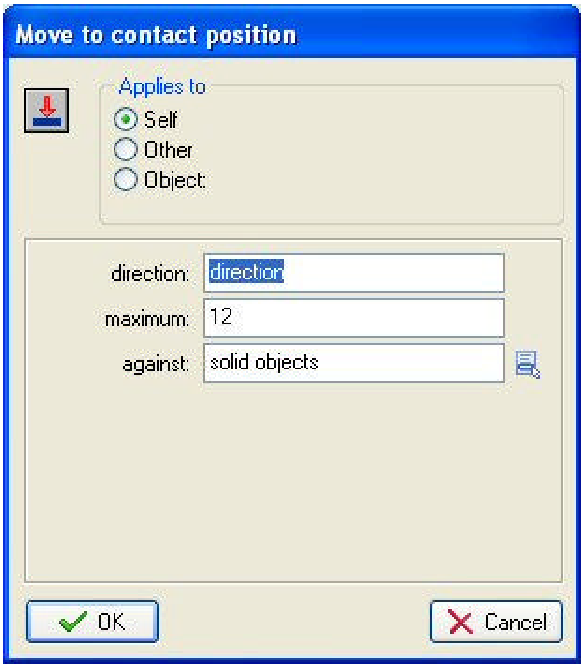
Actions:
- *Check whether in the air*
- If a position is collision free
- Set the gravity
- Else
- Set the gravity
- *Limit the vertical speed*
- If vspeed is larger than 12
- Set variable vspeed to 12

Next we have to land correctly on the floor. This is more difficult than it might seem. It will happen when the character collides with the block object. In this collision event we should set the vertical motion to 0. But this might leave the character hanging a bit in the air above the ground. (The reason is that the character is placed back to its previous position before the collision.) To this end we want to move the character to the exact point where the collision occurs. Fortunately there is action for this in *Game Maker*:

> **Move to contact position**
> With this action you can move the instance in a given direction until a contact position with an object is reached. If there already is a collision at the current position the instance is not

moved. Otherwise, the instance is placed just before a collision occurs. You can specify the direction but also a maximal distance to move. You can also indicate whether to consider solid object only or all objects.

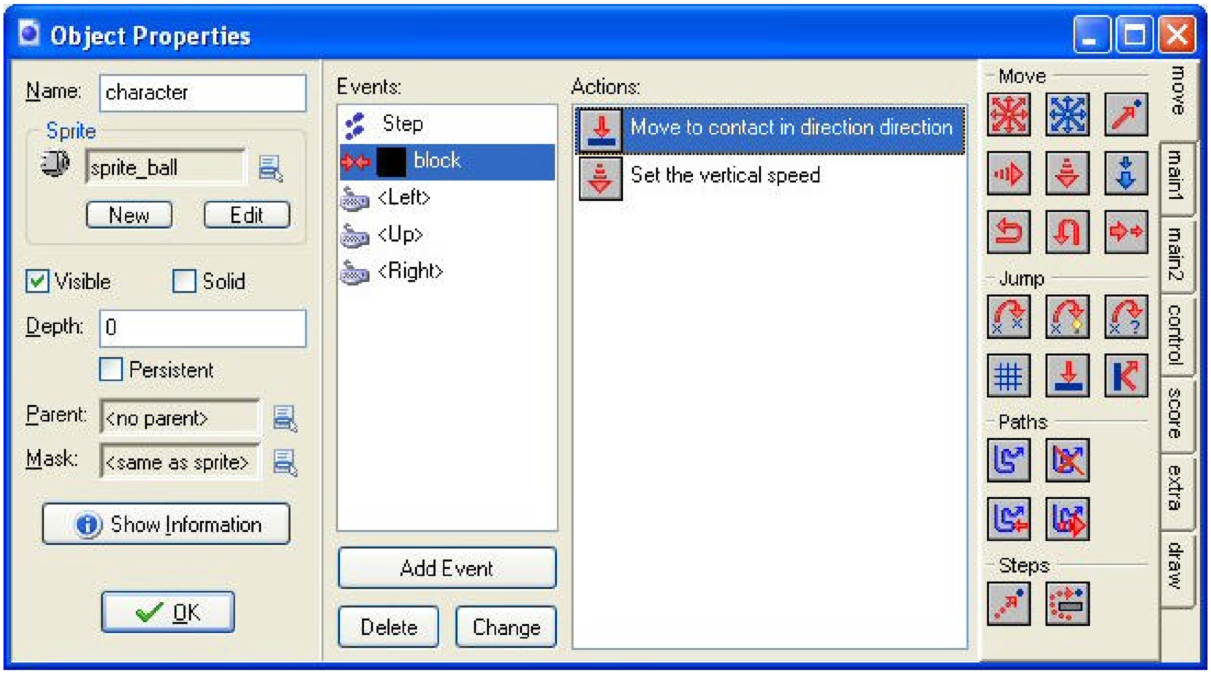
We use this action. As direction we indicate the variable direction which is the current direction of motion of the instance. As a maximal distance we specify 12 (although this is not really necessary here):



● Self
○ Other
○ Object:

direction: direction
maximum: 12
against: solid objects

So the total collision event with the block looks as follows:



Name: character
Sprite: sprite_ball
☑ Visible ☐ Solid
Depth: 0
☐ Persistent
Parent: <no parent>
Mask: <same as sprite>

Events:
- Step
- block
- <Left>
- <Up>
- <Right>

Actions:
- Move to contact in direction direction
- Set the vertical speed

You could argue that we should only do this when we hit a floor below us. But actually we also want to move to the contact position if we hit a floor from below or if we hit a wall from the side. There is one important thing here that is often a cause for problems: We assume that the character at its previous position is indeed collision free. You would expect this but this is not always the case.

A mistake that is often made is that when the character has an animated image, also the collision mask changes in every step. This could mean that the new image at the previous location still causes a collision. So you better make sure that the character has one collision mask (see also in the next section).

Finally we have to let the character jump when the up arrow key is pressed. But this must only happen when the character is currently on the floor. So we first test whether the position below the character creates a collision and, if so, set the vertical speed e.g. to -10. You might have to play a bit with the vale of 10 for the vertical speed and the value of 0.5 for the gravity to get the motion you want.

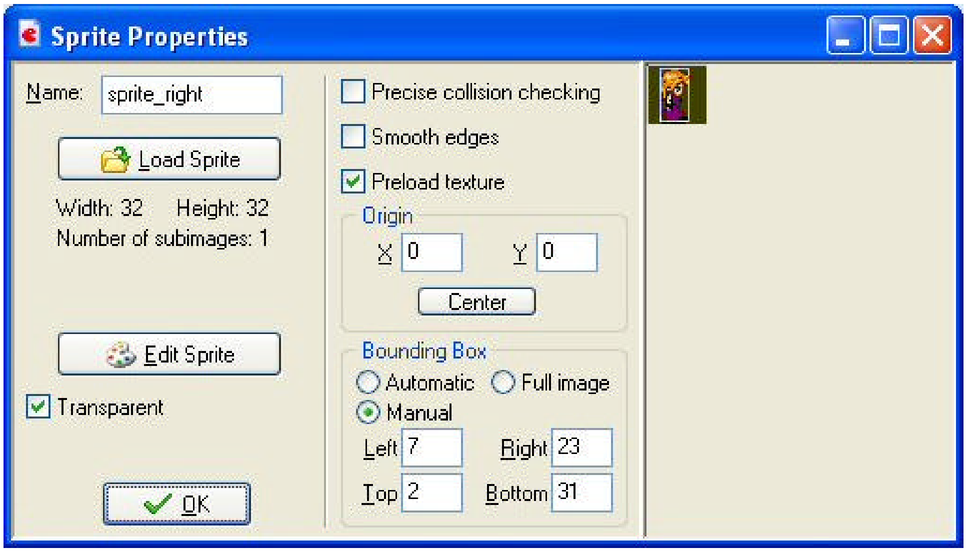
Now the basis for the platform game is ready. Design a level with some floors and walls, constructed from instances of the block object. Place an instance of the character in the room and you are done.

Better graphics

The basic platform game we created in the previous section works but it looks rather bad. There are two aspects we want to change: the way the player looks, and the way the background looks. The adapted game can be found in the file `platform_2.gm6`.

The character images

Let's start with the character graphics. We will use two different (non-animated) sprites: one for the character facing to the left and one for the character facing to the right. The easiest now is to place in the event for the left arrow key an action to change the sprite to the one facing left. Similar, in the right arrow key you switch to the one with the character facing right. It is very important that you switch off precise collision checking for the two sprites. There are a number of reasons for this. First of all, it avoid that the sprite get stuck halfway down the edge of the platform. Secondly, when the sprite is changed from left facing to right facing they should use the same collision mask otherwise the character might get stuck. The same is even more important when using animated sprites. For the same reason you better make sure that the bounding boxes of the sprites are the same. You can always use manual bounding boxes for this. So when adding the sprites the settings should be something like this:



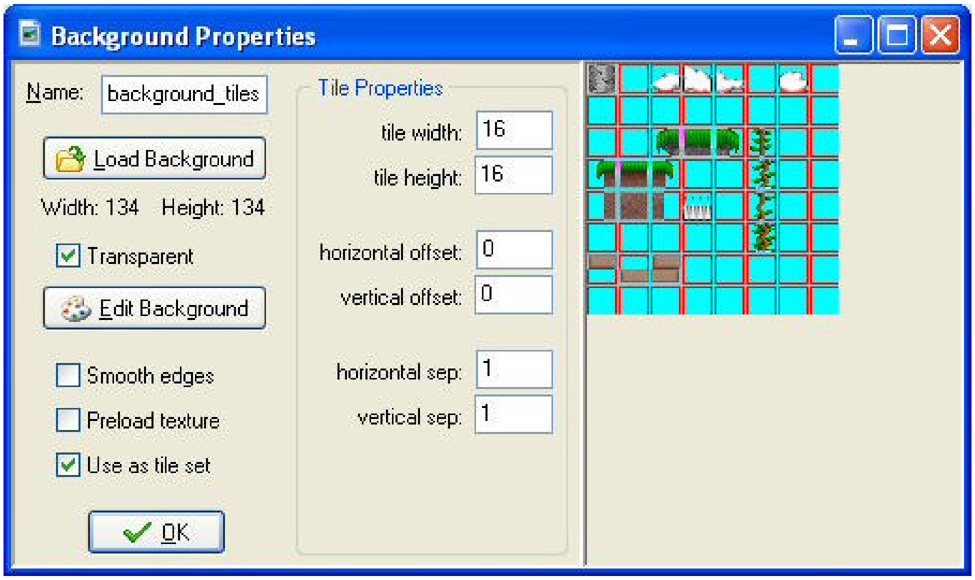
Name: sprite_right
Width: 32 Height: 32
Number of subimages: 1
☑ Transparent
☐ Precise collision checking
☐ Smooth edges
☑ Preload texture
Origin
X 0 Y 0
Bounding Box
○ Automatic ○ Full image
● Manual
Left 7 Right 23
Top 2 Bottom 31

In more advanced games you will probably want to use animated sprites. In this case you also need a sprite for the character when it is not moving. Also you might want to add sprites for the character jumping, falling, shooting, etc. In this case you will have to change the sprite at various places in the events. In particular, in the no key event you probably want to set the sprite to the not moving one. Alternatively, you can draw the correct sprite in the drawing event based on the situation. For example, you can check whether `xprevious<x` to find out whether the character has moved to the right. As I indicated before, better make sure that all sprites have the same bounding box and no precise collision checking.

The platforms and walls

Secondly we want to improve the background and the platforms. Here we use a standard technique. Rather than using objects for all the different wall and floor elements, we use so-called tiles. Tiles are pieces of background images that are drawn at particular places in the room. They do not have associated events nor do they create collision. The good part is that they are fast and use little memory. So you can create large rooms without the need for large images.

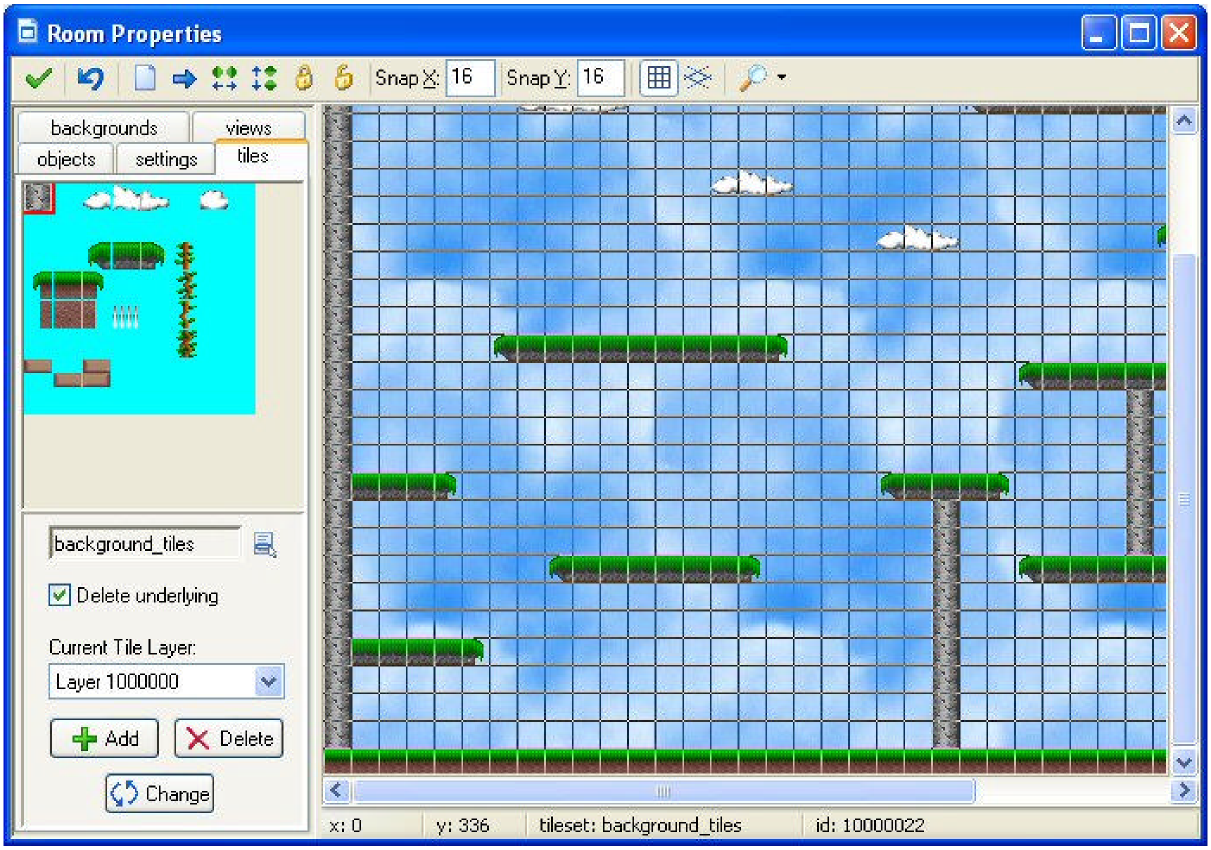
To add tiles to your rooms you first need a background image that contains the tiles. Tiles in a background image preferably have a fixed size and have a little (1-pixel) border between them such that they can easily be separated. For our simple platform game we made our own which is included. We added it as a transparent background resource named `background_tiles`. When adding it to the game, in the background properties, indicate that it should be used as a tile set and fill in the correct tile size and separation, as follows:



Name: background_tiles
Width: 134 Height: 134
☑ Transparent
☐ Smooth edges
☐ Preload texture
☑ Use as tile set

Tile Properties
tile width: 16
tile height: 16
horizontal offset: 0
vertical offset: 0
horizontal sep: 1
vertical sep: 1

Now, when creating a room, you can click on the **Tiles** tab page. You can select the tile set (that is, the appropriate background resource). Now you can draw tiles by clicking on the appropriate tile and next placing them in the room, like you would do for objects. The right mouse button deletes tiles. Use your imagination to create challenging rooms. (Note that you can place tiles on different depth layers by adding layers. For example, you can make a layer of tiles that lie in front of the moving characters. We will not use them here but they are great for giving a better 3D effect.)



Snap X: 16 Snap Y: 16

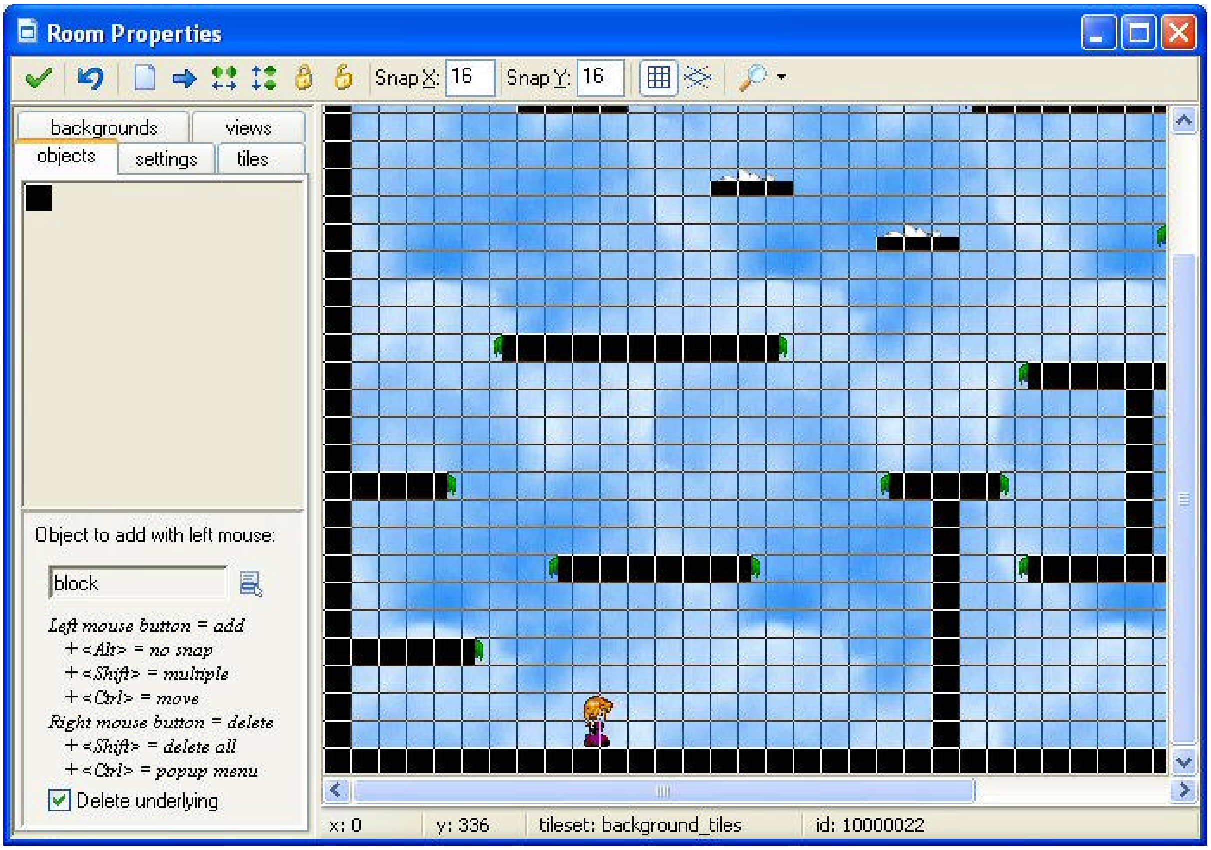
background_tiles

☑ Delete underlying

Current Tile Layer: Layer 1000000

x: 0 y: 336 tileset: background_tiles id: 10000022

There is a problem left though. As indicated above, tiles are just nice graphics. They do not generate events or collisions. So the character would fall straight through them. To avoid this we still need the block objects we had before. We place the block objects at the appropriate places on top of the walls and platforms you did create with the tiles on the background. Now by making the block objects invisible you will not see the black blocks but the beautiful tiles. But the block objects are actually there, so the character cannot pass through the walls and will land on the platforms.



Snap X: 16 Snap Y: 16

Object to add with left mouse:

block

Left mouse button = add
+ <Alt> = no snap
+ <Shift> = multiple
+ <Ctrl> = move
Right mouse button = delete
+ <Shift> = delete all
+ <Ctrl> = popup menu

☑ Delete underlying

x: 0 y: 336 tileset: background_tiles id: 10000022

There might be one problem here. The 16x16 block objects will be too large to cover the background nicely. So we want to make a few other block objects of size 16x8 and 8x16. Again we make them solid. To avoid having to specify collision events with these as well, we use the parent mechanism. This is a very powerful mechanism that you should learn to use. If an object A is a parent of object B, B behaves as a special case of A. It inherits all the behavior of A (unless you overwrite this with other behavior). Also, collisions with B are treated the same as collisions with A. So for the smaller blocks we set the parent to the bigger block. In this way they will be treated the same as the bigger block.

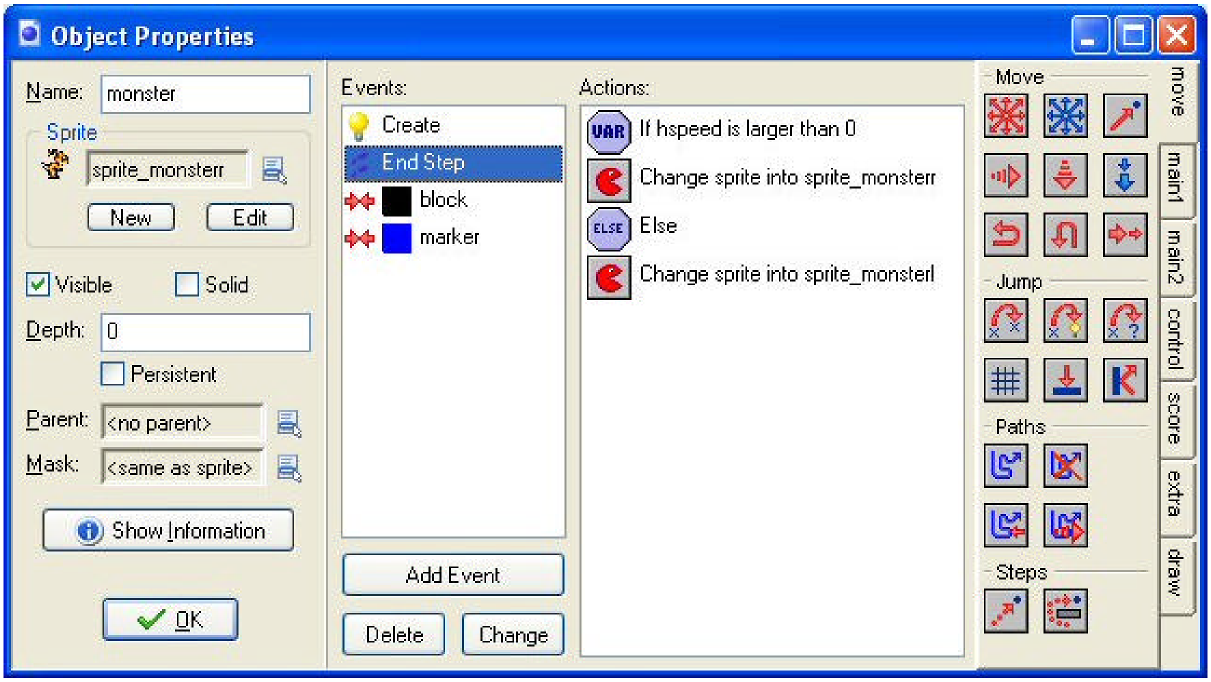
Threats and treats

Just jumping around from platform to platform is rather boring. You definitely need some more challenges and goals. In this section we treat a number of these. Check out the game `platform_3.gm6` for the result.

Monsters

Let us first add some monsters. We will make two monsters, one that moves left and right on a platform and the other that flies left and right in the sky. Jumping on top of it can squash the first one; the second one should be avoided at all times.

Let's start with the monster that moves on the platforms. We need two sprites for it, one with the monster facing left and the other with the monster facing right. Again, better don't use precise collision checking for the same reasons as indicated above and pick some relevant bounding box. Now we create the monster object. In the creation event we let it move to the right with a particular speed. Whenever it hits a wall it reverses its horizontal speed. To set the correct sprite for the monster we use the **end step** event. This event happens just before the instances are drawn. In it we set the correct sprite based on the value of the variable `hspeed` that indicates the horizontal speed. If it is smaller than 0 we let the monster face left and otherwise we let it face right. (If you have the registered version of *Game Maker* you can also use the action to mirror the image. In this case you need only one sprite.)



Name: monster

Sprite: sprite_monsterr

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:

- Create
- End Step
- block
- marker

Actions:

- If hspeed is larger than 0
- Change sprite into sprite_monsterr
- Else
- Change sprite into sprite_monsterl

To avoid monsters from falling off platforms, we introduce another object, which we call a marker. This marker will be an invisible red block. Whenever a monster touches it, it reverses its direction of motion. Having invisible markers is a good general trick to let instances perform certain actions at particular places in your room. Besides changing direction you could use markers to shoot, to lay bombs, etc.

When the character hits a monster, the character should die. But actually, as in most platform games we like to make it possible for the character to jump on top of the monster and squash it. So in the collision event of the character with the monster we must check whether we hit the monster from above to squash it. To find out we perform the following test:

```
vspeed > 0 && y < other.y+8
```

It is true if `vspeed` is larger than 0, so the character moves downwards, and the character is close to the top of the monster so it is indeed hitting it from above In this case the monster must be destroyed. (In the example we turn the monster into a flat dead monster, which destroys itself after a while. This gives a nicer graphical effect.) In this simple platform game, dying for the character corresponds to restarting the level, which can be achieved by some simple actions.

The flying monster is even easier. We proceed in exactly the same way. Only, in the collision event of the character with the flying monster, no test needs to be performed because you cannot squash a flying object.

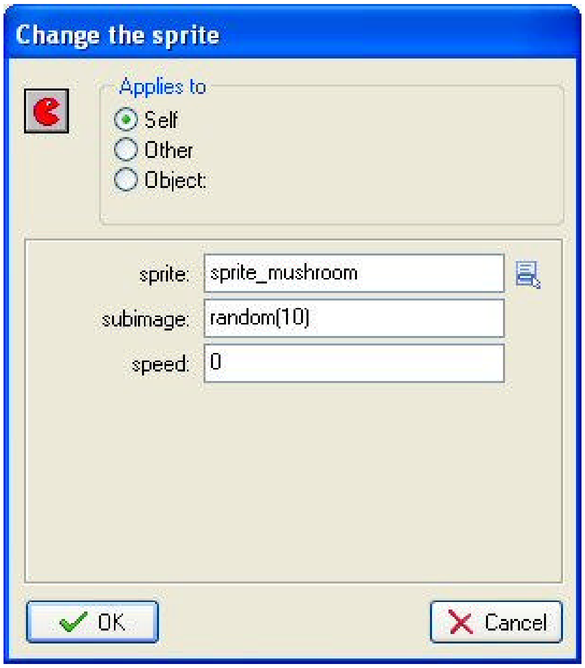
You might want to add some more monsters, e.g. with different speeds, to make things harder. You can also make a monster or rock that falls down or moves up and down. Just use your own imagination.

Pits

Most platform games require careful timing of jumps to avoid falling into pits. Falling into a pit normally kills the character. To this end, we add a new object, called death. This object is a red block that again is not visible. You can place it at the bottom of the pit. (In the tiled room you can put some spikes there.) In the collision event of the character with the death object it should play a sound, wait a while, and restart the room. You can also make pits that go down infinitely. In this case you want to add similar actions in the outside event (in the other events) of the character, maybe including a test whether `y > room_height` to make sure the character fell down, rather than jumped up outside of the playing field.

Collecting points

Most platform games have some mechanism in which the player can collect points. Normally you have to pick up certain objects or catch certain things. In our example the player can collect mushrooms. So we make a mushroom object. To give a bit of variation, the mushroom sprite contains 10 different mushrooms sub-images. The mushroom object picks one at random upon creation using the action to set the sprite:



Applies to
● Self
○ Other
○ Object:

sprite: sprite_mushroom
subimage: random(10)
speed: 0

We set the sub-image to `random(10)`.`random(10)` is a function call. It will return a random number below the argument given (so below 10 in our case). We set the speed to 0 to stop cycling through the subimages. In the collision event of the character with the mushroom object we play a sound, destroy the other object (that is, the mushroom) and add 10 to the score.

In some platform games, collecting things has a more important function than just raising your score. For example, you might get an extra life when you collect enough objects. Also there might be objects that restore your health (assuming monsters don't kill you but simply weaken you), make you move faster, jump higher, etc. These can easily be added.

Next level

Of course there should be a way to finish a level, such that the player can move on to the next level. To this end, we create a `levelexit` object. When the character gets there you are moved to the next level. In the example this is done rather simple. We add a test action to see whether the next room exists. If this test is true we move to the next room. Otherwise the highscore list is shown and the game is restarted.

You might choose to make the level exit only appear when for example all mushrooms have been collected. To this end, in the creation event of the `levelexit` object, move it to a position –100,-100 (so off the screen). Now in the step event of the object we check whether the number of mushroom objects is equal to 0 (there is an action for this) and, if so, move the object back to its starting position (again there is an action for this). All very simple.

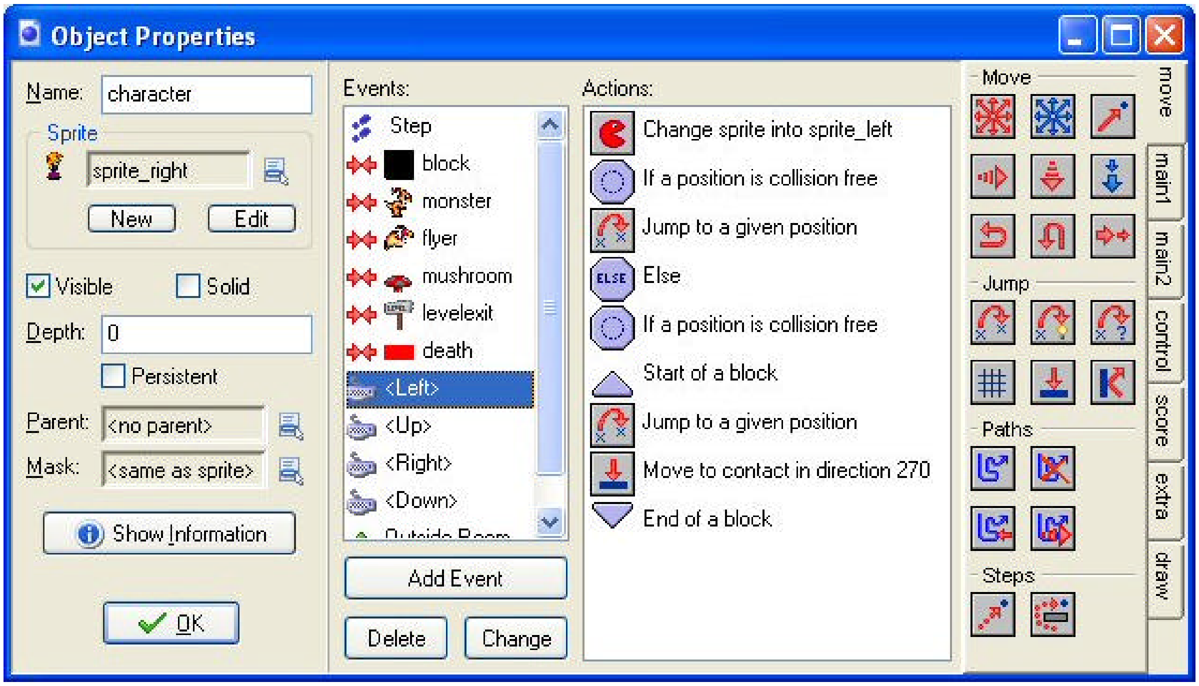
More motions

Our current platform game has just some limited motion possibilities. The character can move left and right, and it can jump. To make things more interesting, let us add some possibilities. The result can be found in the game `platform_4.gm6`.

Ramps

It is nice if the player can walk up sloping ramps (down goes automaticaly because of the falling). To this end, we have to replace the code in the left arrow key event. We put there the following:

Rather than just testing whether the position to the left is collision free we also test whether a position 8 pixels higher is collision free. If so we move the character there and use the landing action to move it down to the contact position. So the event will look as follows:



Name: character

Sprite: sprite_right

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Change sprite into sprite_left
If a position is collision free
Jump to a given position
Else
If a position is collision free
Start of a block
Jump to a given position
Move to contact in direction 270
End of a block

The right arrow key is handled in a similar way.

Ladders

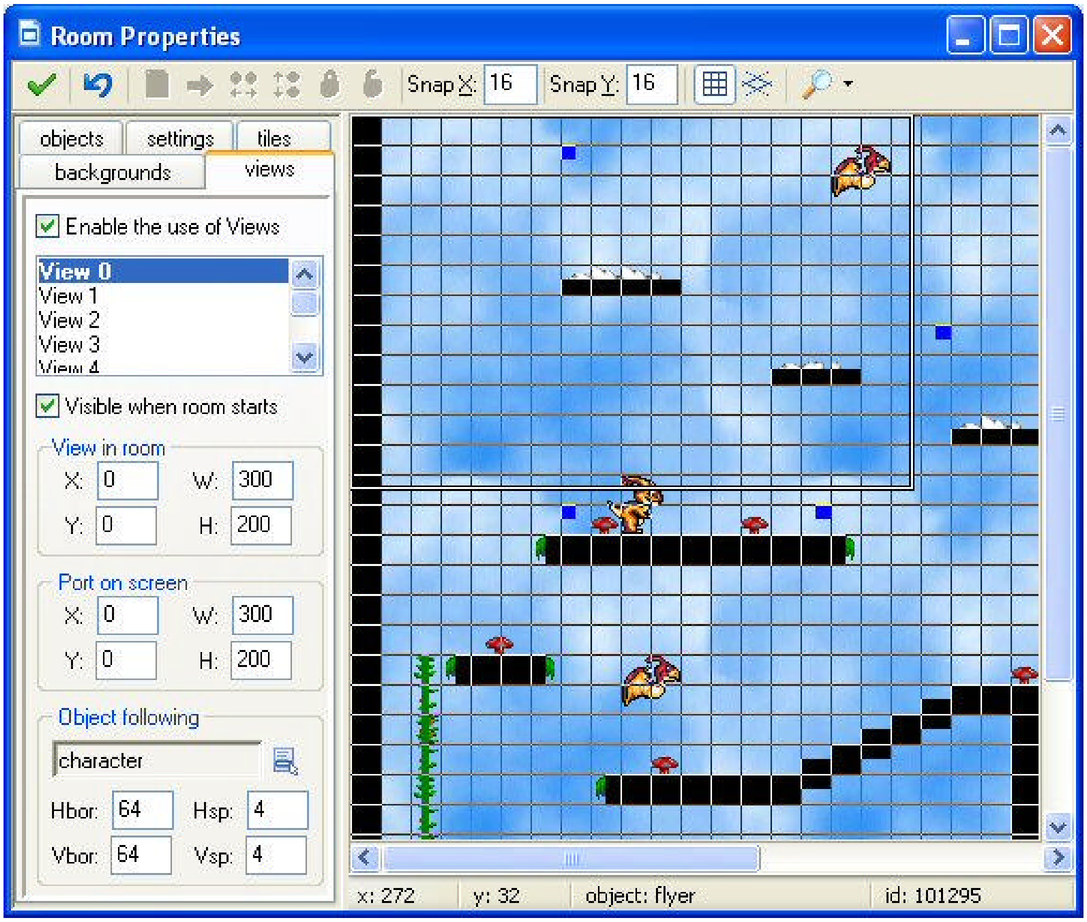
People always want ladders in platform game along which the character can move from one platform to the other. This requires a bit of work. A ladder will be represented by a thin vertical block that is invisible (the real ladder or vine or whatever that is used for climbing is drawn again using tiles) and not solid. When the character is not in contact with a ladder, motion should be as before. But when it is in contact with the ladder things must go different. First of all, the character should not fall down. So in the step event we have to make a change to this effect adding some actions that set the vertical speed and gravity to 0 when in contact with a ladder. Also we set the sprite to the climbing sprite in that case.

The second thing that needs to change is the event for the up key. When the character is at a ladder, the up arrow key should move it up, rather than jump. Again we need a few additional actions for this. We test whether the character is in contact with a ladder and, if so, move it up a bit. We use similar actions for the down key.

Using a view

Up to now we always showed the entire room. For many platform games this is not what you want. Instead you want to see only a part of the room, around the character you are controlling. This makes the game more challenging because the player must try to detect his way through the platforms. You can also hide prizes at difficult to reach places in the room.

Fortunately this is extremely simple to achieve in *Game Maker*. When designing the room, click on the **Views** tab. Click on the checkbox **Enable the use of Views** to start using views. Select the first view and check the box **Visible when room starts** to make sure this view can be seen. Give it a width of 300 and a height of 200 (or something else that you like). (As we are going to let the view follow the character there is no need to specify the left and top position of the view in the room. Also, because we use just one view, we don't have to specify the x and y position of the port on the screen.) At the bottom we can indicate which object to follow. Here we choose the character. The view will now automatically move to keep the character in focus. We don't want the character to get too close to the border. To this end we set the **Hbor** and **Vbor** values to 64. There will now always be a 64 pixel area visible around the character. Finally, to get a smooth view motion we set the maximal view speed to 4. (This also gives a very nice effect at the start because the character comes slowly into view.) So the settings will look as follows:



Snap X: 16
Snap Y: 16

☑ Enable the use of Views

☑ Visible when room starts

View in room
X: 0 W: 300
Y: 0 H: 200

Port on screen
X: 0 W: 300
Y: 0 H: 200

Object following
character
Hbor: 64 Hsp: 4
Vbor: 64 Vsp: 4

x: 272 y: 32 object: flyer id: 101295

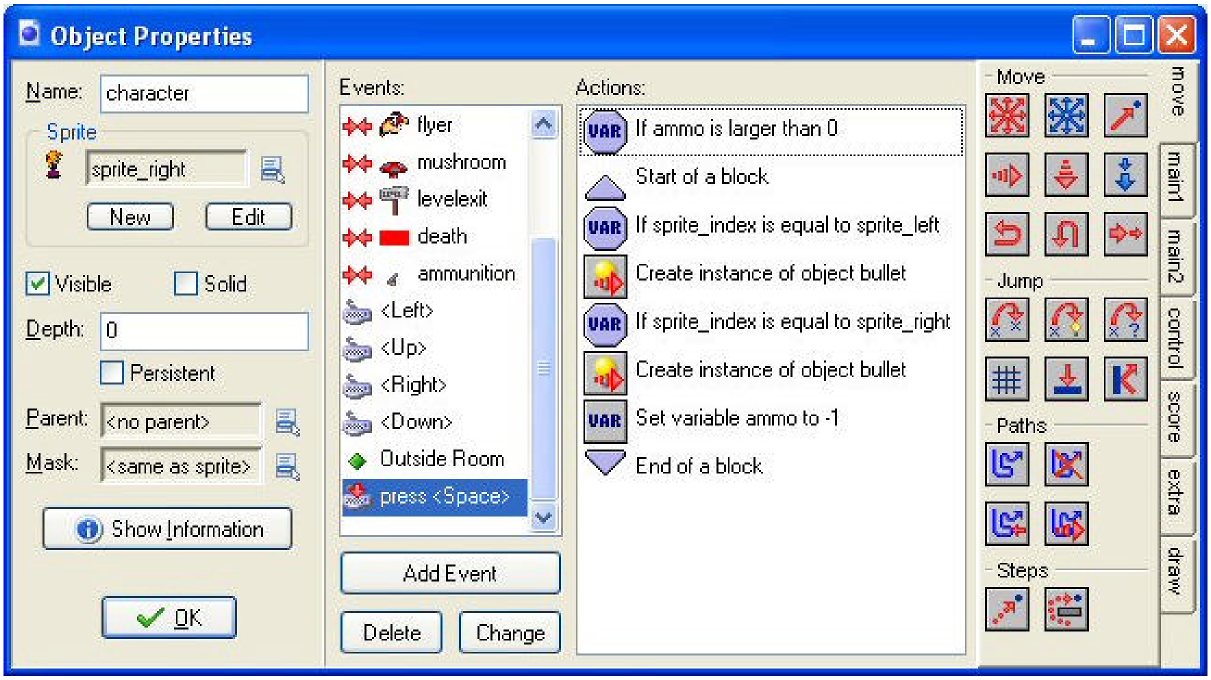
Having the view is nice but it makes the window in which things happen rather small. To avoid this, in the **General Game Settings** we indicate a fixed scale of 200 percent. Clearly you can play with these values to get the effect you want.

Some further touches

Shooting monsters

The next step is to enable the player to shoot monsters. To make things a bit more interesting, the player first needs to find some ammunition to be able to shoot. To this end we introduce a variable we call `ammo` that indicates how much ammunition the player has. In the creation event of the character we set this to 0 using the action to set a variable. The ammunition object has a simple sprite and does nothing. It just waits to be picked up by the player. When the character collides with the ammunition object we add 10 to the variable `ammo` (set it relative to 10) and destroy the ammunition instance.

Next we need a bullet object When the player presses the <Space> key an instance of this object must created, assuming there is ammunition, and the value of the variable `ammo` is decreased by 1. But there is one important issue. We like the bullet to shoot in the direction the character is facing. To this end we check the value of variable `sprite_index`. This variable contains the index of the sprite for the character object. Based on it we create a bullet with the correct direction of motion. When we are climbing no bullet is created. (Shooting while climbing is not possible.) So the space event looks as follows:



Name: character

Sprite: sprite_right

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events:
- flyer
- mushroom
- levelexit
- death
- ammunition
- <Left>
- <Up>
- <Right>
- <Down>
- Outside Room
- press <Space>

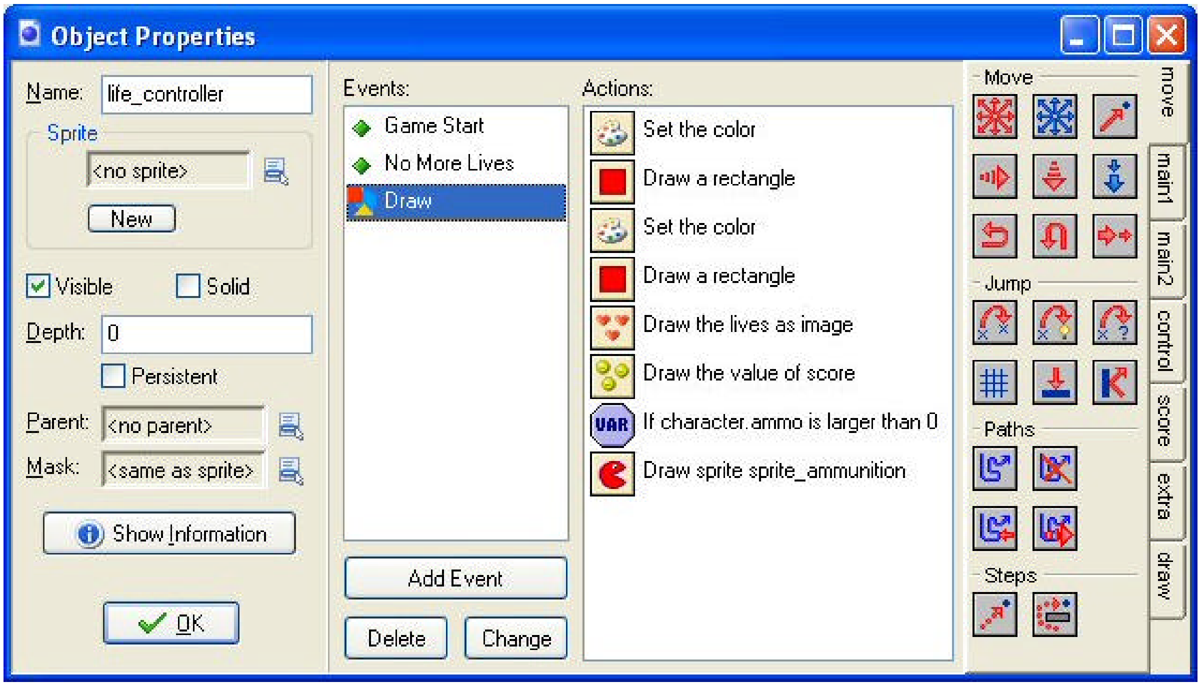
Actions:
- If ammo is larger than 0
- Start of a block
- If sprite_index is equal to sprite_left
- Create instance of object bullet
- If sprite_index is equal to sprite_right
- Create instance of object bullet
- Set variable ammo to -1
- End of a block

It remains to destroy the bullet when it hits a wall or when it goes outside the room and to kill the monster when the bullet hits it. This is all easy. See the file `platform_5.gm6` for the new version of the game.

A score panel

A player now has a score and ammunition. We are also going to give it some lives. Hitting a monster or falling in a pit will cost a life. There is an easy mechanism for lives in *Game Maker*. We are going to create a special `life_controller` object. It does not need a sprite. In its creation event it sets the number of lives to 3. Whenever the player dies we decrease the number of lives. In the **No more lives** event for the controller we show the highscore table and restart the game.

But it would also be nice if we can see the number of lives, the score, ammunition, etc. To this end we are going to make a little panel with this information. We are going to draw this in the **Draw** event of the controller object. There is though a problem here. Where should we draw it? We cannot draw it at a fixed place in the room because the view changes and we want the panel always in view. Fortunately we can ask for the position of the view. This is indicated by the two variables `view_xview` and `view_yview` that indicate the left position and top position of the view respectively. So we can draw the panel with the information relative to this position. Here is what the draw event of the controller object looks like:



Name: life_controller

Sprite: <no sprite>

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

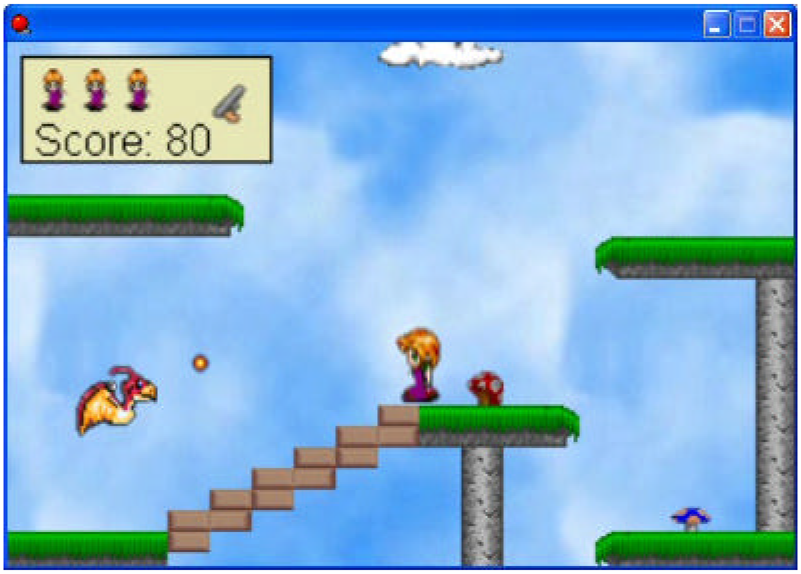
Parent: <no parent>

Mask: <same as sprite>

Events:
- Game Start
- No More Lives
- Draw

Actions:
- Set the color
- Draw a rectangle
- Set the color
- Draw a rectangle
- Draw the lives as image
- Draw the value of score
- If character.ammo is larger than 0
- Draw sprite sprite_ammunition

Note that we also draw an image when the player can shoot. In the game, that can be found in the file `platform_6.gm6`, this result in the following image:



What next?

The sections above have explained some of the basics of making platform games. Now it is your turn. You will have to use these techniques and some more ideas of yourself to create a real nice platform game. Remember that the most crucial part of platform games is formed by the levels. Start making levels one by one. Play them until you are happy with them. Every so often, introduce some new game play aspect. Here are some additional ideas that you can use:

- different monsters, e.g. bouncing balls and monsters that shoot
- keys that you need to find in order to open doors
- mines that you can place somewhere and that go off when a monster (or yourself) steps on them
- water to swim in (this will completely change the motions; no gravity anymore, or a mild upwards gravity until you reach the surface, limited time before you run out of air, air bubbles to grab, etc.)
- walls and floors you can destroy, e.g. by shooting them or jumping on them with force
- trampolines that make you jump higher
- platforms that appear and disappear
- one-way streets
- moving platforms (this is not easy!)
- …

Good luck.